APPENDIX

An exemplary interface definition illustrating code that defines an API according to an embodiment of the present invention is presented below. As noted above, it will be appreciated that any such definition is equally compatible with an embodiment of the present invention, and therefore the present invention is not limited to the language, text, format, and the like, of the below-listed exemplary interface definition. The comments included in the definition are for illustrative purposes only, and are not a comprehensive listing of the functions that may be performed by an embodiment of the present invention.

```
// Type definitions
typedef DWORD PID;
typedef DWORD SDI_SPID;
typedef DWORD BID;
typedef DWORD WID;
typedef DWORD IDX;
typedef DWORD OFF;
typedef DWORD PRID;
typedef DWORD DBGDBID;
typedef UINT64 TASKID;
typedef DWORD NLVL;

// Forward declarations of interfaces
interface IHostDebugDebuggerInstance;
interface IHostDebugServerInstance;
interface IHostDebugEvent;
interface IHostDebug;

// Interface definitions
[
  uuid(87bc18db-c8b3-11d5-ae96-00b0d0e93cc1),
  pointer_default(unique)
]
interface IHostDebugDebuggerInstance : IUnknown
{
  HRESULT OnMatch(
    [in, string] const wchar_t * szClientMachine,
    [in] PID pidClient,
    [in] IHostDebug * pDebug,
    [out] IHostDebugEvent ** ppDebugEvent);
}

[
  uuid(87bc18dc-c8b3-11d5-ae96-00b0d0e93cc1),
  pointer_default(unique)
]
interface IHostDebugServerInstance : IUnknown
{
  HRESULT Register(
    [in, string] const wchar_t * szClientMachine,
    [in] PID pidClient,
    [in] IHostDebugDebuggerInstance * pInstance,
    [in] BOOL fReservedMustBeFalse, // DebugExistingConnections
    [out] DWORD * pdwCookie,
```

```
    [out] DWORD * pdwPidServer);
  HRESULT Unregister(
    [in] DWORD dwCookie);
}

typedef struct HOST_DEBUG_SPAN
{
  IDX    m_idxStart;
  OFF    m_offStart;
  OFF    m_offEnd;
} HOST_DEBUG_SPAN;

typedef struct HOST_DEBUG_STATEMENT
{
  HOST_DEBUG_SPAN m_span;
        BOOL m_bDynamic;
        [switch_is(m_bDynamic)] union
        {
                [case(TRUE)] struct
                {
                        BID m_bid;
                        WID m_wid;
                        NLVL m_nlvl;
                };
                [case(FALSE)] struct
                {
                        PRID    m_prid;
                        [string] wchar_t * m_szDbName;
                        DBGDBID    m_dbid;
                };
        };
} HOST_DEBUG_STATEMENT;

typedef enum HOST_DEBUG_STEPKIND
{
        HOST_DEBUG_STEPKIND_STEPCOMPLETE = 1,
        HOST_DEBUG_STEPKIND_BREAKPOINT_HIT = 2,
        HOST_DEBUG_STEPKIND_ASYNC_BREAK = 8
} HOST_DEBUG_STEPKIND;

// This interface is implemented by the debugger and called by SQL SERVER
[
  uuid(87bc18dd-c8b3-11d5-ae96-00b0d0e93cc1),
  pointer_default(unique)
]
interface IHostDebugEvent : IUnknown
{
  // Called on new connection to the database
  HRESULT NewConnection(
    [in] SDI_SPID spidParent,
    [in] BID bidParent,
    [in] WID widParent,
    [in] SDI_SPID spid,
    [in, string] const wchar_t * szSqlServer,
    [in, string] const wchar_t * szMachine,
    [in, string] const wchar_t * szClientMachine,
    [in, string] const wchar_t * szClientProcessName,
    [in, string] const wchar_t * szClientUserName,
    [in] PID pid,
    [in] DWORD dwThreadId);
  // Called when a new batch is started (MARS).
  // There is an implicit worker
```

```
// (WID==0) that is assumed with this operation
// (ie. NewWorker is not
// called unless a UDF actually gets parallelized).
HRESULT NewBatch(
   [in] BID bid);
// Called for each helper that is created during
// parallel execution of a UDF
HRESULT NewWorker(
   [in] BID bid,
   [in] WID wid,
   [in] NLVL nlvl);
// Called when a stored procedure is executed.
HRESULT Push(
   [in] BID bid,
   [in] WID wid,
   [in, string] const wchar_t * szDBName,
   [in, string] const wchar_t * szObjectName,
   [in] DBGDBID dbid,
   [in] PRID prid,
   [in] NLVL nlvl,
   [in] UINT_PTR esp,
   [in] ULONG ulThreadId,
   [in] USHORT usValidSpans,
   [in, size_is(usValidSpans)] HOST_DEBUG_SPAN * arDebugSpan);
HRESULT PushDynamic(
   [in] BID bid,
   [in] WID wid,
   [in] NLVL nlvl,
   [in] UINT_PTR esp,
   [in] ULONG ulThreadId,
   [in, string] wchar_t * szText,
   [in] USHORT usValidSpans,
   [in, size_is(usValidSpans)] HOST_DEBUG_SPAN * arDebugSpan);
HRESULT Step(
   [in] BID bid,
   [in] WID wid,
   [in] PRID prid,
   [in] NLVL nlvl,
   [in] HOST_DEBUG_STEPKIND stepkind,
   [in] HOST_DEBUG_SPAN span,
   [in] ULONG ulThreadId);
// Balances out calls to NewSP
HRESULT Pop(
   [in] BID bid,
   [in] WID wid,
   [in] NLVL nlvl);
HRESULT CloseConnection([in] SDI_SPID spid);
HRESULT CloseBatch(
   [in] BID bid);
HRESULT CloseWorker(
   [in] BID bid,
   [in] WID wid);
HRESULT Trace(
   [in] BID bid,
   [in] WID wid,
   [in, string] const wchar_t * szOutput);
HRESULT CallManagedFromTsql(
   [in] BID bid,
   [in] WID wid,
   [in] TASKID taskid);
HRESULT ReturnToTsqlFromManaged(
   [in] BID bid,
```

```
        [in] WID wid,
        [in] TASKID taskid);
    HRESULT CallTsqlFromManaged(
        [in] BID bid,
        [in] WID wid,
        [in] TASKID taskid);
    HRESULT ReturnToManagedFromTsql(
        [in] BID bid,
        [in] WID wid,
        [in] TASKID taskid);
    HRESULT Ping();
}

interface IEnumHostDebugSymbol;

[
        uuid(87bc18df-c8b3-11d5-ae96-00b0d0e93cc1),
        pointer_default(unique)
]
interface IHostDebugBinary : IUnknown
{
        HRESULT Get(
                ULARGE_INTEGER ulOffset,
                ULONG cbToRead,
                [out, size_is(cbToRead),
                        length_is(*pcbActuallyRead)] char * arcBuf,
                [out] ULONG * pcbActuallyRead);
        HRESULT Set(
                [in] ULONG cbWrite,
                [in, size_is(cbWrite)] char * arcBuf);
        HRESULT GetSize(
                [out] DWORD * pdwSize);
}

typedef enum HOST_DEBUG_SYMBOL_TYPE
{
        HOST_DEBUG_SYMBOL_TYPE_LEAF,
        HOST_DEBUG_SYMBOL_TYPE_COMPOUND,
        HOST_DEBUG_SYMBOL_TYPE_BINARY
} HOST_DEBUG_SYMBOL_TYPE;

typedef enum HOST_DEBUG_SYMBOL_PROPERTIES
{
    HOST_DEBUG_SYMBOL_PROPERTIES_GLOBAL = 1,
    HOST_DEBUG_SYMBOL_PROPERTIES_LOCAL = 2,
    HOST_DEBUG_SYMBOL_PROPERTIES_PARAM = 4,
    HOST_DEBUG_SYMBOL_PROPERTIES_VAR_NA1 = 8,
    HOST_DEBUG_SYMBOL_PROPERTIES_VAR_NA2 = 16,
    HOST_DEBUG_SYMBOL_PROPERTIES_READONLY = 32,
    HOST_DEBUG_SYMBOL_PROPERTIES_NULL = 64,
    HOST_DEBUG_SYMBOL_PROPERTIES_XML = 128,
        HOST_DEBUG_SYMBOL_PROPERTIES_TEXT = 256
} HOST_DEBUG_SYMBOL_PROPERTIES;

// Represents a variable in SQL SERVER.
// Allows simple types, array, and heirarchy
typedef struct HOST_DEBUG_SYMBOL
{
    [string] wchar_t * m_szName;
    [string] wchar_t * m_szType;
    ULONG m_ulProperties;
    HOST_DEBUG_SYMBOL_TYPE m_symbolType;
```

```
    [switch_is(m_symbolType)] union
    {
      [case(HOST_DEBUG_SYMBOL_TYPE_LEAF)] struct
      {
        [string] wchar_t * m_szValue;
      };
      [case(HOST_DEBUG_SYMBOL_TYPE_COMPOUND)] struct
      {
                        IEnumHostDebugSymbol * m_pEnum;
      };
                [case(HOST_DEBUG_SYMBOL_TYPE_BINARY)] struct
                {
                        IHostDebugBinary * m_pBinary;
                };
    };
} HOST_DEBUG_SYMBOL;

[
        uuid(87bc18e0-c8b3-11d5-ae96-00b0d0e93cc1),
        pointer_default(unique)
]
interface IEnumHostDebugSymbol : IUnknown
{
        HRESULT Next(
                [in] ULONG ulSymbolsToFetch,
                [out, size_is(ulSymbolsToFetch), length_is(*pulSymbolsFetched)]
HOST_DEBUG_SYMBOL * arSymbols,
                [out] ULONG * pulSymbolsFetched);
        HRESULT Skip(
                [in] ULONG ulSymbolsToSkip);
        HRESULT Reset();
        HRESULT Clone(
                IEnumHostDebugSymbol ** ppEnum);
        HRESULT GetCount(
                [out] DWORD * pdwCount);
}

typedef enum HOST_DEBUG_STEP_MODE
{
  HOST_DEBUG_STEP_INTO,
  HOST_DEBUG_STEP_OVER,
  HOST_DEBUG_STEP_TORETURN,
  HOST_DEBUG_STEP_GO
} HOST_DEBUG_STEP_MODE;

[
  uuid(87bc18de-c8b3-11d5-ae96-00b0d0e93cc1),
  pointer_default(unique)
]
interface IHostDebug : IUnknown
{
  HRESULT Continue(
    [in] BID bid,
    [in] WID wid,
    [in] HOST_DEBUG_STEP_MODE step);
  HRESULT StopDebugEvents();
  HRESULT SetBreakpoints(
    [in] BOOL fEnable,
    [in] long nBreakpoints,
    [in, size_is(nBreakpoints)] HOST_DEBUG_STATEMENT * arBreakpoints);
  HRESULT GetCallstack(
    [in] BID bid,
```

```
        [in] WID wid,
        [out] long * pnStatements,
        [out, size_is(, *pnStatements)] HOST_DEBUG_STATEMENT ** parStatements);
    HRESULT AsyncBreak(
        [in] BID bid,
        [in] WID wid);
    HRESULT RevokeBreak(
        [in] BID bid,
        [in] WID wid);
    HRESULT SetSym(
        [in] BID bid,
        [in] WID wid,
        [in] NLVL nlvl,
        [in] HOST_DEBUG_SYMBOL * pSymbol);
    HRESULT GetSyms(
        [in] BID bid,
        [in] WID wid,
        [in] NLVL nlvl,
        [out] USHORT * pusSymbols,
        [out, size_is(,*pusSymbols)] HOST_DEBUG_SYMBOL ** ppSymbols);
    HRESULT GetGlobalSym(
        [in] BID bid,
        [in] WID wid,
        [in, string] wchar_t * szVariable,
        [out] HOST_DEBUG_SYMBOL * pSymbol);
    HRESULT LookupTaskIdentifier(
        [in] BID bid,
        [in] WID wid,
        [out] TASKID * pTaskid);
    // If the debugger gets hosed,
    // allow the server to continue untouched
    HRESULT StopDebugging();
    HRESULT CreateManagedDebuggerHost(
                [in] wchar_t * szVersion,
                        // version string passed from debugger
                [in] wchar_t * szArg);
                        // argument string passed from debugger
    // This is disabled by default
    HRESULT ToggleManaged(
        [in] BOOL fEnableManaged);
        HRESULT Ping();
}
```